Bits & Bots

CSE 3241

Sam Bossley | Michael Izzo | Josephine Ko | Henry Xiong

Part I

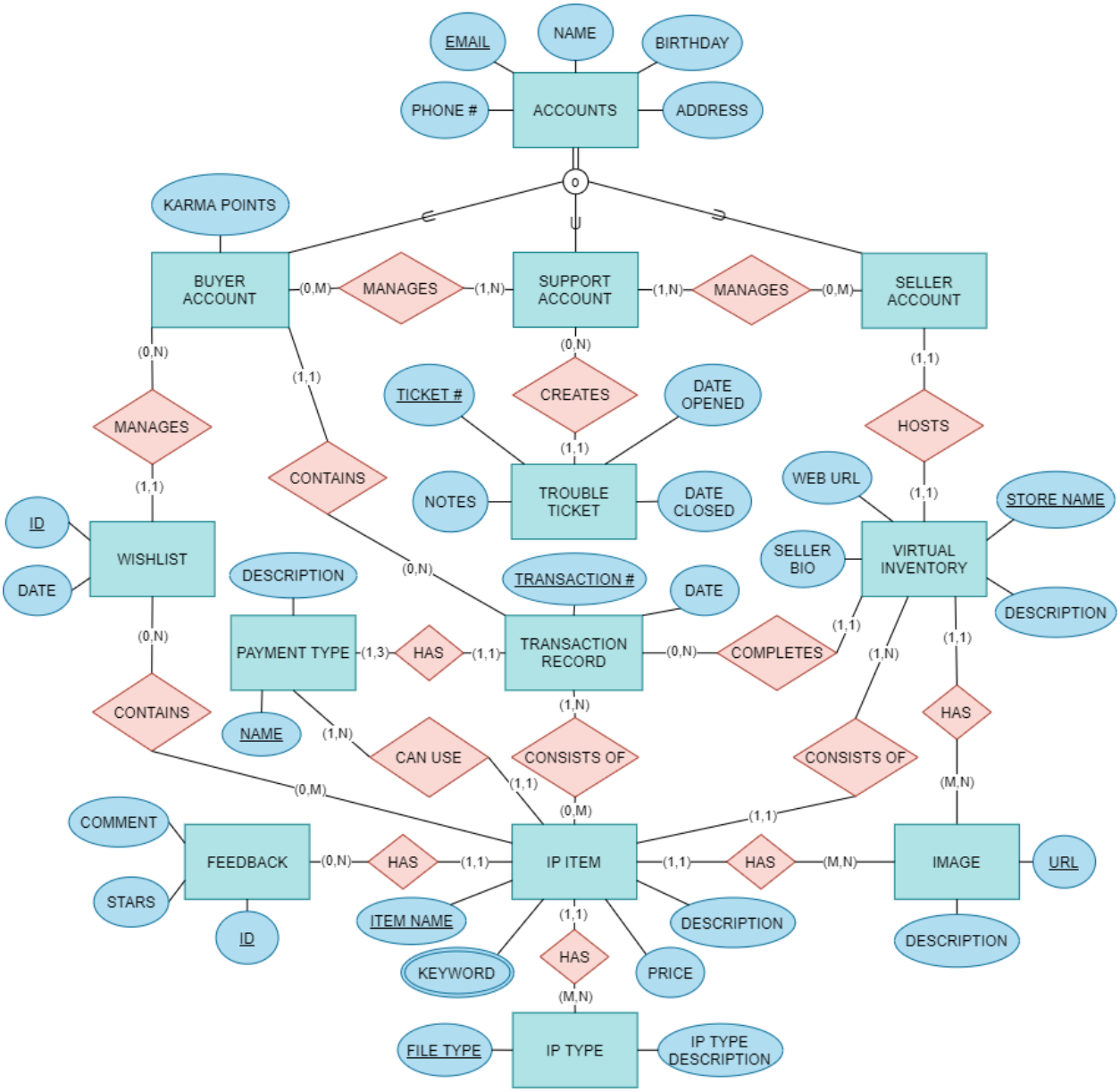
Section I - Database Description

Assignment Task:
You and your project team are employed by DB 4Ever., a consulting company with clients worldwide. You've been assigned to help Ms Yotta Bietz set up a database for her latest entrepreneurial enterprise, BITS & BOTS. The application will be a kind of online marketplace for the maker community. It will permit makers to set up small virtual storefronts for securely distributing intellectual property, collecting payments and interacting with users.

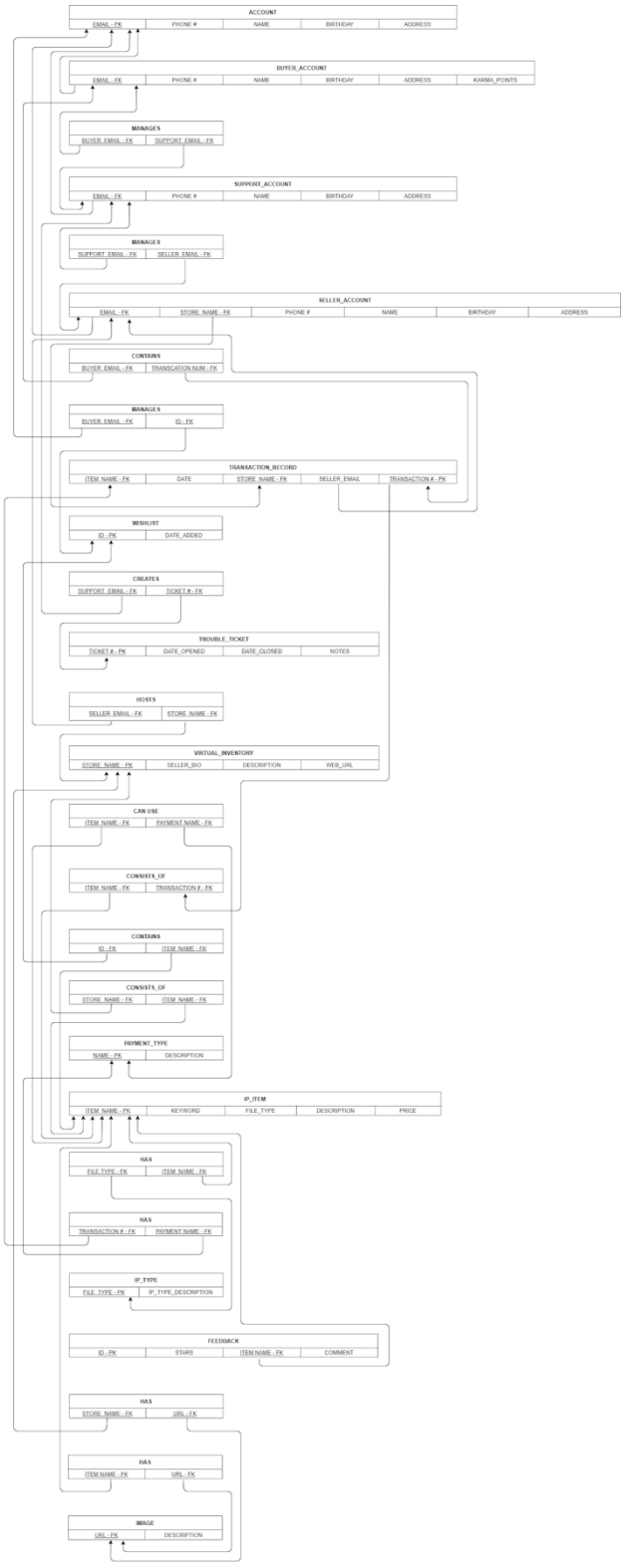
Ms. Bietz needs a simple information management system and database to support virtual inventory, buyer/seller accounts, and sales operations. You have been given some sample data and test scenarios. You will need to supplement your solution with features that you consider important or interesting.

Entity Relationship Diagram

The relational schema diagram consists of several main parts that tie to each other. The first part is the Accounts superclass that the entities Buyer Account, Support Account and Seller Account inherits. The main structure that needs to be remembered between the three is that Support Accounts manage both Buyer Accounts and Seller Accounts. Meanwhile, the second major section is the different miscellaneous things that the different accounts are tied to. For example, Buyer Account is tied to Wishlist, and Seller Account is tied to Virtual Inventory, while Support Accounts are tied to Trouble Tickets. Lastly, the third and final major section is concerning items in the database in general and how to organize them as well as the information behind them. This includes things like how every IP Item is tied to an Image entity, a Feedback entity and of course has ties in with a specific IP Type.



Relational Schema



Normalization

Normalization is the process of "normalizing" lower normal forms to high normal forms through methods such as elimination or alteration of functional dependencies, or through decomposition, or in some cases, alteration of the very structure of the data itself.

Every single functional dependency's normalization and converted form (into 3NF at least specifically) is listed below. If the dependency was originally in a lower level normal form, it is also listed prior. Every single dependency that is listed as 3NF is also BCNF.

ACCOUNT
R = { Email, Phone_num, Name, Birthday, Address }
3NF (BCNF) = { Email → Phone_num, Name, Birthday, Address }
This table has reached 3NF because it is not only in 2NF, but there is no non-key attached to another non-key, therefore it is in 3NF.

BUYER_ACCOUNT
R = { Email, Phone_num, Name, Birthday, Address }
3NF (BCNF) = { Email → Phone_num, Name, Birthday, Address }
This table has reached 3NF because it is not only in 2NF, but there is no non-key attached to another non-key, therefore it is in 3NF.

SUPPORT_MANAGES_BUYER
R = { Buyer_email (FK), Support_email (FK) }
No dependencies, 3NF (BCNF)
This table has reached 3NF because it is not only in 2NF automatically, but there is also no non-key in the first place to prevent 3NF, so it is in 3NF.

SUPPORT_ACCOUNT
R = {Email, Phone_num, Name, Birthday, Address }
3NF (BCNF) = {Email → Phone_num, Name, Birthday, Address }
This table has reached 3NF because it is not only in 2NF, but there is no non-key attached to another non-key, therefore it is in 3NF.

SUPPORT_MANAGES_SELLER
R = { Support_email (FK), Seller_email (FK) }
No dependencies, 3NF (BCNF)
This table has reached 3NF because it is not only in 2NF automatically, but there is also no non-key in the first place to prevent 3NF, so it is in 3NF.

SELLER_ACCOUNT
R = { Email, Phone_num, Name, Birthday, Address }
3NF (BCNF) = { Email → Phone_num, Name, Birthday, Address }

This table has reached 3NF because it is not only in 2NF, but there is no non-key attached to another non-key, therefore it is in 3NF.

BUYER_CONTAINS_TRANSACTION_RECORDS
R = { Buyer_email (FK), Transaction_num (FK) }
No dependencies, 3NF (BCNF)
This table has reached 3NF because it is not only in 2NF automatically, but there is also no non-key in the first place to prevent 3NF, so it is in 3NF.

BUYER_MANAGES_WISHLIST
R = { Buyer_email (FK), Id (FK) }
No dependencies, 3NF (BCNF)
This table has reached 3NF because it is not only in 2NF automatically, but there is also no non-key in the first place to prevent 3NF, so it is in 3NF.

TRANSACTION_RECORD
R= { Transaction_num, Date, Store_name (FK), Item_name (FK) }
1NF = { Transaction_num → Date, Store_name (FK), Item_name (FK) }
3NF (BCNF) = { Transaction_num, Store_name (FK), Item_name (FK) → Date }
This table has reached 3NF because it is not only in 2NF, but there is no non-key attached to another non-key, therefore it is in 3NF.

WISHLIST
R = { Id, Date_added }
3NF (BCNF) = { Id → Date_added }
This table has reached 3NF because it is not only in 2NF, but there is no non-key attached to another non-key, therefore it is in 3NF.

SPT_ACCT_CREATES_TICKET
R = { Support_email (FK), Ticket_num (FK) }
No dependencies, 3NF (BCNF)
This table has reached 3NF because it is not only in 2NF automatically, but there is also no non-key in the first place to prevent 3NF, so it is in 3NF.

TROUBLE_TICKET
R = { Ticket_num, Date_opened, Date_closed, Notes }
3NF (BCNF) = { Ticket_num → Date_opened, Date_closed, Notes }
This table has reached 3NF because it is not only in 2NF, but there is no non-key attached to another non-key, therefore it is in 3NF.

SELL_ACCT_HOSTS_INV
R = { Store_name (FK), Seller_email (FK) }
No dependencies, 3NF (BCNF)

This table has reached 3NF because it is not only in 2NF automatically, but there is also no non-key in the first place to prevent 3NF, so it is in 3NF.

VIRTUAL_INVENTORY
R = { Store_name, Seller_bio, Description, Web_url }
3NF (BCNF) = { Store_name → Seller_bio, Description, Web_url }
This table has reached 3NF because it is not only in 2NF, but there is no non-key attached to another non-key, therefore it is in 3NF.

ITEM_CANUSE_PAYMENT_TYPE
R = { Item_name (FK), Payment_name (FK) }
No dependencies, 3NF (BCNF)
This table has reached 3NF because it is not only in 2NF automatically, but there is also no non-key in the first place to prevent 3NF, so it is in 3NF.

TRANSACTION_RECORDS_CONSISTOF_IP_ITEMS
R = { Item_name (FK), Transaction_num (FK) }
No dependencies, 3NF (BCNF)
This table has reached 3NF because it is not only in 2NF automatically, but there is also no non-key in the first place to prevent 3NF, so it is in 3NF.

WISHLIST_CONTAINS_IP_ITEM
R = { Id (FK), Item_name (FK) }
No dependencies, 3NF (BCNF)
This table has reached 3NF because it is not only in 2NF automatically, but there is also no non-key in the first place to prevent 3NF, so it is in 3NF.

INV_CONSISTSOF_IP_ITEMS
R = { Store_name (FK), Item_name (FK) }
No dependencies, 3NF (BCNF)
This table has reached 3NF because it is not only in 2NF automatically, but there is also no non-key in the first place to prevent 3NF, so it is in 3NF.

PAYMENT_TYPE
R = { Name, Description }
3NF (BCNF) = { Name → Description }
This table has reached 3NF because it is not only in 2NF, but there is no non-key attached to another non-key, therefore it is in 3NF.

IP_ITEM
R = { Item_name, Keyword, File_type, Description, Price }
3NF (BCNF) = { Item_name → Keyword, File_type, Description, Price }
This table has reached 3NF because it is not only in 2NF, but there is no non-key attached to another non-key, therefore it is in 3NF.

IP_ITEM_HAS_IP_TYPE
R = { File_type (FK), Item_name (FK) }
No dependencies, 3NF (BCNF)
This table has reached 3NF because it is not only in 2NF automatically, but there is also no non-key in the first place to prevent 3NF, so it is in 3NF.

TRANSACTION_RECORD_HAS_PAYMENT_TYPE
R = { Transaction_num (FK), Payment_name (FK) }
No dependencies, 3NF (BCNF)
This table has reached 3NF because it is not only in 2NF automatically, but there is also no non-key in the first place to prevent 3NF, so it is in 3NF.

IP_TYPE
R = { File_type, IP_type_description }
3NF (BCNF) = { File_type → IP_type_description }
This table has reached 3NF because it is not only in 2NF, but there is no non-key attached to another non-key, therefore it is in 3NF.

FEEDBACK
R = { Id, Stars, Item_name (FK), Comment }
1NF = { Id → Stars, Item_name (FK), Comment }
3NF (BCNF) = { Id, Item_name (FK) → Comment, Stars }
This table has reached 3NF because it is not only in 2NF, but there is no non-key attached to another non-key, therefore it is in 3NF.

INV_HAS_IMAGES
R = { Store_name (FK), Url (FK) }
No dependencies, 3NF (BCNF)
This table has reached 3NF because it is not only in 2NF automatically, but there is also no non-key in the first place to prevent 3NF, so it is in 3NF.

IP_ITEM_HAS_IMAGES
R = { Item_name (FK), Url (FK) }
No dependencies, 3NF (BCNF)
This table has reached 3NF because it is not only in 2NF automatically, but there is also no non-key in the first place to prevent 3NF, so it is in 3NF.

IMAGE
R = { Url, Description }
3NF (BCNF) = { Url → Description }
This table has reached 3NF because it is not only in 2NF, but there is no non-key attached to another non-key, therefore it is in 3NF.

Indices

Indices, or indexes, are a data structure made to complement databases because searching through a large database with a regular query without any indices takes too long. Indices can consist of anything from tree structures to hash table structures that would enable queries to be faster by tying itself as references to certain areas of the database so that a query is essentially able to "skip" to that area instead of going through every database row.

Our two indices that we have suggested are:

1. The first index is indexing for a specific store name. This is helpful because users of the database can find the exact store name they are looking for quickly. Hashing would be the most appropriate type of indexing because it is the quickest in regard to using equalities.

   Code:
   ```
   CREATE [UNIQUE] INDEX Store_Name_Index
   ON Virtual_Inventory(Store_Name);
   ```

2. The second index is indexing for an IP item with specifications on the cost range. This is helpful because users can identify certain items within the range of cost they have determined they want to spend. The best type of indexing for this would be B-Tree, this is the fastest way to find all of the values the user is asking for.

   Code:
   ```
   CREATE INDEX IP_Item_Cost_Index
   ON IP_Item(Price);
   ```

Views

Feature below are two sample views which will likely be frequently used in the implemented database.

The first view finds each store's most expensive listing. This view is useful to consumers (or buyers) to determine which store sells the cheapest items overall, and it is useful for sellers to be able to determine how their prices compare with their competitors. It additionally may be useful for Bits & Bots to gather information on the highest priced listings of every store for research and data gathering purposes.

The relational algebra for the aforementioned view is as follows:

$$\text{VIRT\_STORE} \leftarrow (\text{VIRTUAL\_INVENTORY}) \bowtie_{\text{VIRTUAL\_INVENTORY.Store\_name=VIRTUAL\_INVENTORY\_CONSISTSOF\_IP\_ITEMS.Store\_name}} (\text{VIRTUAL\_INVENTORY\_CONTSISTSOF\_IP\_ITEMS})$$

$$\text{ALL\_ITEMS} \leftarrow (\text{VIRT\_STORE}) \bowtie_{\text{VIRT\_STORE.Item\_name=IP\_ITEM.Item\_name}} (\text{IP\_ITEM})$$

$$\Im_{\text{MAX(PRICE)}} (\pi_{\text{Store\_name, Item\_name, Price}} \text{ALL\_ITEMS})$$

The corresponding SQL code is below.

```
CREATE VIEW STORES_MOST_EXPENSIVE_LISTING
        SELECT I.Item_name, MAX(I.Price), S.Store_name
        FROM VIRTUAL_INVENTORY AS S, IP_Item AS I,
VIRTUAL_INVENTORY_CONSISTSOF_IP_ITEMS as V
        WHERE V.Store_name = S.Store_name
                AND V.Item_name = I.Item_name
        ORDER_BY S.Store_name
```

This view might produce data similar to the data below.

STORES_MOST_EXPENSIVE_LISTING

Store_name	**Item_name**	**Price**
Microsoft	Microsoft 365	499.99
Apple	MacOS Virus	329.99
PrivacyRespecters	AdBlock 2.0	180.00
Google Store	Google One	210.00
DuoMobile	DuoMobile	1000.00
Playstation	Playstation Online	40.00
Xbox	Xbox Live	38.00
Youtube	Youtube Premium	599.99

The second view takes a more practical route and finds all accounts that a specific support account has helped manage. This information can be extremely helpful to trace the history of all the accounts a support account has helped, or if a certain support account has already helped a given buyer or seller.

The relational algebra is below. In this example, we are making the assumption that the support account email is “smithybob@rocketmail.com”.

SUPPORTED_SELLERS ← (SUPPORT_MANAGES_SELLER) ⋈

SUPPORT_MANAGES_SELLER.Support_email=”smithybob@rocketmail.com” AND SUPPORT_MANAGES_SELLER.Seller_email=SELLER.email
(SELLER)

SUPPORTED_BUYERS ← (SUPPORT_MANAGES_BUYER) ⋈

SUPPORT_MANAGES_BUYER.Support_email=”smithybob@rocketmail.com” AND SUPPORT_MANAGES_BUYER.Buyer_email=BUYER.email
(BUYER)

SUPPORTED_ACCTS ← ($\pi_{\text{Buyer\_email, Seller\_email}}$ (SUPPORTED_BUYERS) ⋈
SUPPORT_MANAGES_BUYER.Support_email=”smithybob@rocketmail.com” AND
SUPPORTED_BUYERS.Support_email=SUPPORTED_SELLERS.Support_email (SUPPORTED_SELLERS))

The corresponding SQL code is below.

```
CREATE VIEW ALL_MANAGED_ACCOUNTS
        SELECT Seller_email, Buyer_email
        FROM SUPPORT_MANAGES_SELLER as S, SUPPORT_MANAGES_BUYER as B
        WHERE B.Support_email = ‘smithybob@rocketmail.com’
        OR S.Support_email = ‘smithybob@rocketmail.com’;
```

This view might produce data like this:

ALL_MANAGED_ACCOUNTS (Support_email=”smithybob@rocketmail.com”)

Buyer_email	**Seller_email**
jim.kirk@gmail.com	NULL
NULL	bryson@gmail.com
NULL	google.account@gmail.com
john.doe@gmail.com	NULL
tikewon.doe@gmail.com	NULL
NULL	stop.asking@yahoo.com
NULL	dw@wiki.org

Transactions

Below are two sample database transactions that might prove helpful when querying the database for particular data.

The first transaction removes an item (IP_ITEM) from a given store’s virtual inventory (VIRTUAL_INVENTORY), where both the store name and item name are uniquely identifiable. This is a proper delete operation. In this example, we use “Google Store” to represent the store name, and “Virus.exe” to represent the item to be removed from that store.

```
BEGIN TRANSACTION REMOVE_IP_ITEM

        CREATE VIEW ALL_IP_ITEMS
        SELECT Store_name, Item_name
        FROM VIRTUAL_INVENTORY AS V
```

```
        INV_CONSISTSOF_IP_ITEMS AS C,
        IP_ITEM AS I
    WHERE V.Store_name = “Google Store”
        AND V.Store_name = C.Store_name
        AND C.Item_name = I.Item_name

    IF error THEN GO TO UNDO; END IF;

        DELETE FROM ALL_IP_ITEMS
        WHERE I.Item_name =”Virus.exe”;
        IF error THEN GO TO UNDO; END IF;

        COMMIT;
        GO TO FINISH;
    UNDO:
        ROLLBACK;
    FINISH:

END TRANSACTION;
```

The second transaction adds an item (IP_ITEM) to a buyer’s (BUYER_ACCOUNT) wishlist (WISHLIST). This is a standard insert operation. In this example, we will make the assumption that the buyer is uniquely identifiable by “smithybob@rocketmail.com”, and the item is identified by “Virus.exe”.

```
BEGIN TRANSACTION WISHLIST_ADD_ITEM

    CREATE VIEW WISHLIST_TOTAL_LIST
    SELECT Buyer_email, Item_name
    FROM BUYER_MANAGES_WISHLIST AS M,
        WISHLIST_CONTAINS_IP_ITEM AS W
    WHERE M.Id = W.Id
        AND M.Buyer_email = ‘smithybob@rocketmail.com’;

        IF error THEN GO TO UNDO; END IF;

        INSERT INTO WISHLIST_TOTAL_LIST VALUES (
            ‘smithybob@rocketmail.com’,
            ‘Virus.exe’
        );

        IF error THEN GO TO UNDO; END IF;

    COMMIT;
    GO TO FINISH;
```

```
        UNDO:
                ROLLBACK;
        FINISH:
END TRANSACTION;
```

Section II - User Manual

We have created a user manual for our database containing descriptors for each relation, including attribute information and additional sample queries. Listed below are the aforementioned relations:

- ACCOUNT - This relation is a superclass which represents a general account. It encompasses buyer accounts, seller accounts, and support accounts.
    - Email - a primary key representing the email address for the account (string)
    - Phone_num - the account phone number (10 digit integer)
    - Name - this pertains to the name given to the account - usually this is the user's name (string)
    - Birthday - the birthday of the user creating the account, to verify age restrictions (date)
    - Address - the address of the user creating the account, for identification/billing purposes (string)
- BUYER_ACCOUNT - This relation is a specialization of ACCOUNT which represents the accounts of all customers.
    - Email - a primary key representing the email address for the account (string)
    - Phone_num - the account phone number (10 digit integer)
    - Name - this pertains to the name given to the account - usually this is the user's name (string)
    - Birthday - the birthday of the user creating the account, to verify age restrictions (date)
    - Address - the address of the user creating the account, for identification/billing purposes (string)
- SUPPORT_MANAGES_BUYER - The relation represents the support accounts managing the buyer accounts.
    - Buyer_email (FK) - a foreign key representing the email address for the buyer account (string)
    - Support_email (FK) - a foreign key representing the email address for the support account (string)
- SUPPORT_ACCOUNT - This relation is a specialization of ACCOUNT which represents the accounts of all support members.
    - Email - a primary key representing the email address for the account (string)
    - Phone_num - the account phone number (10 digit integer)
    - Name - this pertains to the name given to the account - usually this is the user's name (string)
    - Birthday - the birthday of the user creating the account, to verify age restrictions (date)
    - Address - the address of the user creating the account, for identification/billing purposes (string)
- SUPPORT_MANAGES_SELLER - This relation represents the support accounts managing the seller accounts.
    - Support_email (FK) - a foreign key representing the email address for the support account (string)
    - Seller_email (FK) - a foreign key representing the email address for the seller account (string)

- SELLER_ACCOUNT - This relation is a specialization of ACCOUNT which represents the accounts of all sellers owning a virtual store.
    - Email - a primary key representing the email address for the account (string)
    - Phone_num - the account phone number (10 digit integer)
    - Name - this pertains to the name given to the account - usually this is the user's name (string)
    - Birthday - the birthday of the user creating the account, to verify age restrictions (date)
    - Address - the address of the user creating the account, for identification/billing purposes (string)
- BUYER_CONTAINS_TRANSACTION_RECORDS - This relation represents all transactions that have transpired, and to which buyer they belong.
    - Buyer_email (FK) - a foreign key representing the email address for the buyer account (string)
    - Transaction_num (FK) - a foreign key used to identify a particular transaction (integer)
- BUYER_MANAGES_WISHLIST - This relation represents all wishlists and to which buyer they belong.
    - Buyer_email (FK) - a foreign key representing the email address for the buyer account (string)
    - Id (FK) - a foriegn key used to identify particular item(s) within the wishlist (string)
- TRANSACTION_RECORD - This relation represents a purchase transaction and information pertaining to a specific purchase order.
    - Transaction_num - a primary key used to identify a particular transaction (integer)
    - Date - the date the transaction took place, used for analytical purposes (date)
    - Store_name (FK) - this foriegn key pertains to the name given to the account, used to identify the virtual inventory (string)
    - Item_name (FK) - this foreign key pertains to the name given to a particular item, used to identify the item (string)
- WISHLIST - This relation represents a buyer's wishlist, containing items and the date each item was added to the list
    - Id - a primary key used to identify particular item(s) within the wishlist (string)
    - Date_added - the date a particular item was added into the wishlist (date)
- SPT_ACCT_CREATES_TICKET - This relation represents the tickets created by support accounts. It connects individual tickets to the support account by which it was created.
    - Support_email (FK) - a foreign key representing the email address for the support account (string)
    - Ticket_num (FK) - a foreign key that is used to identify the ticket (integer)
- TROUBLE_TICKET - This relation represents a ticket created by a support account pertaining to a particular service request or issue.
    - Ticket_num - a primary key that is used to identify the ticket (integer)
    - Date_opened - the date the ticket was first opened (date)
    - Date_closed - the date the ticket was last closed (date)
    - Notes - comments left to document information relevant to the ticket (string)
- SELL_ACCT_HOSTS_INV - This relation connects a seller account to their virtual inventory. It simply connects the corresponding pairs in a table.

    - Store_name (FK) - this foreign key pertains to the name given to the account, used to identify the virtual inventory (string)
    - Seller_email (FK) - a foreign key representing the email address for the seller account (string)
- VIRTUAL_INVENTORY - This relation represents the store of a seller, including the seller information and the products the seller might be selling.
    - Store_name - this primary key pertains to the name given to the account, used to identify the virtual inventory (string)
    - Seller_bio - a short description of the seller; an introduction (string)
    - Description - a brief explanation of the store, which may include what they sell, where they source from, their mission, etc (string)
    - Web_url - the web address associated with this particular store (string)
- ITEM_CANUSE_PAYMENT_TYPE - This relation specifies what payment methods are acceptable for each ip item.
    - Item_name (FK) - this foreign key pertains to the name given to a particular item, used to identify the item (string)
    - Payment_name (FK) - this foriegn key pertains to the name given to the payment method (string)
- TRANSACTION_RECORDS_CONSISTOF_IP_ITEMS - This relation specifies the specific ip items that correspond to each transaction.
    - Item_name (FK) - this foreign key pertains to the name given to a particular item, used to identify the item (string)
    - Transaction_num (FK) - a foreign key used to identify a particular transaction (integer)
- WISHLIST_CONTAINS_IP_ITEM - This relation specifies the specific ip items that are contained within a given wishlist.
    - Id (FK) - a foreign key used to identify particular item(s) within the wishlist (string)
    - Item_name (FK) - this foreign key pertains to the name given to a particular item, used to identify the item (string)
- INV_CONSISTSOF_IP_ITEMS - This relation specifies the individual ip items that are present in a virtual inventory. In particular, this relation defines the items that a seller is selling.
    - Store_name (FK) - this foreign key pertains to the name given to the account, used to identify the virtual inventory (string)
    - Item_name (FK) - this foreign key pertains to the name given to a particular item, used to identify the item (string)
- PAYMENT_TYPE - This relation represents all possible forms of payment that a buyer might use to purchase a particular ip item.
    - Name - this primary key pertains to the name given to the payment method (string)
    - Description - a brief explanation of the payment method (string)
- IP_ITEM - This relation represents an item for sale on a particular seller's inventory. This includes the item name, a keyword representing the item, an item description, and the item price.
    - Item_name - this primary key pertains to the name given to a particular item, used to identify the item (string)
    - Keyword - specific words that are relevant to the particular item so that it is easier to find the item or recommend it as a similar product (string)

    - Description - a short blurb giving relevant information, such as specs, about the item (string)
    - Price - how much money the item costs (positive integer)
- IP_ITEM_HAS_IP_TYPE - This relation connects an ip item to the type of item. For example, an executable might have type ‘exe’. In this case, the ip item type is a separate relation.
    - File_type (FK) - a foreign key identifying the format in which the item is in, used to check compatibility (string)
    - Item_name (FK) - this foreign key pertains to the name given to a particular item, used to identify the item (string)
- TRANSACTION_RECORD_HAS_PAYMENT_TYPE - This relation specifies what payment type was used for each buyer transaction.
    - Transaction_num (FK) - a foreign key used to identify a particular transaction (string)
    - Payment_name (FK) - this foriegn key pertains to the name given to the payment method (string)
- IP_TYPE - An ip type represents the category of item to which an ip item can belong to. An example of this is ‘exe’.
    - File_type - a primary key denoting the format in which the item is in, used to check compatibility (string)
    - IP_type_description - a brief explanation of the file type (string)
- FEEDBACK - This relation represents a text comment or review pertaining to a particular item.
    - Id - a primary key used to uniquely identify a particular feedback (string)
    - Stars - a rating indicating comparative performance of a particular item (integer)
    - Item_name (FK) - this foreign key pertains to the name given to a particular item, used to identify the item (string)
    - Comment - additional information the user would like to provide about an item
- INV_HAS_IMAGES - This relation represents the images that correspond to a particular seller’s inventory. This might include images such as a profile picture, a banner image, and more.
    - Store_name (FK) - this foreign key pertains to the name given to the account, used to identify the virtual inventory (string)
    - Url (FK) - a foreign key containing the web address associated with the image (string)
- IP_ITEM_HAS_IMAGES - This relation represents the images that correspond to a particular seller’s inventory. This is similar to product images on ecommerce sites.
    - Item_name (FK) - this foreign key pertains to the name given to a particular item, used to identify the item (string)
    - Url (FK) - a foreign key containing the web address associated with the image (string)
- IMAGE - This relation represents an image that can be displayed in an online format. This includes a public image url in addition to a brief description for accessibility purposes.
    - Url - a primary key containing the web address associated with the image (string)
    - Description - a brief explanation of the image (string)

Below are some sample queries that may be used in the database.

To find all IP Items by a given seller that costs less than $10 (assuming the seller is identified by email “smithybob@rocketmail.com”):

```
SELECT Item_name
FROM IP_ITEM AS A, VIRTUAL_INVENTORY AS B, INV_CONSISTSOF_IP_ITEMS AS C,
SELL_ACCT_HOSTS_INV AS D
WHERE D.Seller_email = “smithybob@rocketmail.com”
AND D.Store_name = B.Store_name
AND B.Store_name = C.Store_name
AND C.Item_name = A.Item_name
AND A.Price < 10.00;
```

To give all IP Items and their purchase date for a given buyer (where the buyer is given by “smithybob@rocketmail.com”):

```
SELECT a.Item_name, a.Date
FROM TRANSACTION_RECORD AS a, BUYER_CONTAINS_TRANSACTION_RECORDS AS b
WHERE b.Buyer_email = “smithybob@rocketmail.com”
AND b.Transaction_num = a.Transaction_num;
```

To find the seller names for all sellers with less than 5 IP Items for sale:

```
SELECT a.Seller_email
FROM SELL_ACCT_HOSTS_INV AS a, VIRTUAL_INVENTORY AS b, IP_ITEM AS c,
INV_CONSISTSOF_IP_ITEMS AS d
WHERE Count(c.Item_name) < 5
AND c.Item_name = d.Item_name
AND d.Store_name = b.Store_name
AND b.Store_name = a.Store_name;
```

To display all the buyers who purchased a IP Item by a given seller and the names of the IP Items they purchased (where the seller is identified by the unique email “smithybob@rocketmail.com”):

```
SELECT g.Buyer_email, c.Item_name
FROM SELL_ACCT_HOSTS_INV AS a, VIRTUAL_INVENTORY AS b,
INV_CONSISTSOF_IP_ITEMS AS c, IP_ITEM AS d,
TRANSACTION_RECORDS_CONSISTOF_IP_ITEMS AS e, TRANSACTION_RECORDS AS f,
BUYER_CONTAINS_TRANSACTION_RECORDS AS g
WHERE a.Seller_email = “smithybob@rocketmail.com”
AND a.Store_name = b.Store_name
AND b.Store_name = c.Store_name
AND c.Item_name = d.Item_name
AND d.Item_name = e.Item_name
AND e.Transaction_num = f.Transaction_num
AND f.Transaction_num = g.Transaction_num;
```

To find the total number of IP Items purchased by a single buyer (where the buyer is identified by the unique email “smithybob@rocketmail.com”):

```
SELECT Count(a.Item_name)
FROM TRANSACTION_RECORD AS a, BUYER_CONTAINS_TRANSACTION_RECORDS AS b
WHERE b.Buyer_email = "smithybob@rocketmail.com"
AND b.Transaction_num = a.Transaction_num;
```

To find the buyer who has purchased the most IP Items and the total number of IP Items they have purchased:

```
CREATE VIEW BUYER_ITEM_BOUGHT
SELECT b.Buyer_email, Count(a.Transaction_num) AS Buyer_item_count
FROM TRANSACTION_RECORD AS a, BUYER_CONTAINS_TRANSACTION_RECORDS AS b
WHERE b.Transaction_num = a.Transaction_num;

SELECT *
FROM BUYER_ITEM_BOUGHT
WHERE MAX(Buyer_item_count);
```

To get a listing of all buyer accounts and seller accounts:

```
SELECT a.Email, b.Email
FROM SELLER_ACCOUNT AS a, BUYER_ACCOUNT AS b;
```

To find a virtual store's most expensive listing:

Let the given Virtual Inventory be identified by the unique name 'Google Store'

```
SELECT Item_name, MAX(b.Price)
FROM VIRUAL_INVENTORY AS a, IP_Item AS b
WHERE a. Store_name = 'Google Store'
AND a.Item_name = b.Item_name;
```

To find all accounts that a support account has managed:

Let the given Support Account be identified by the unique email 'smithybob@rocketmail.com'

```
SELECT Seller_email, Buyer_email
FROM SELLERS_MANAGED, BUYERS_MANAGED
WHERE Seller_email = 'smithybob@rocketmail.com'
OR Buyer_email = 'smithybob@rocketmail.com';
```

To provide a list of buyer names, along with the total dollar amount each buyer has spent:

```
CREATE VIEW BUYERS_TOTALS(Buyer_name, Buyer_email, Transaction_total)
SELECT a.Name, a.Email, SUM(d.Price)
FROM BUYER_ACCOUNT AS a, BUYER_CONTAINS_TRANSACTION_RECORDS AS b,
TRANSACTION_RECORDS_CONSISTOF_IP_ITEMS AS c, IP_ITEM AS d
WHERE a.Email = b.Buyer_email
```

```
AND b.Transaction_num = c.Transaction_num
AND c.item_name = d.Item_name
GROUP BY a.Email;

SELECT Buyer_name, Transaction_total
FROM BUYER_BUYERS_TOTALS;
```

To provide a list of buyer names and email addresses for buyers who have spent more than the average buyer:

```
SELECT Buyer_name, Buyer_email
FROM BUYER_BUYERS_TOTALS
WHERE Total_spendings > AVG(Total_spendings);
```

To provide a list of the IP Item names and associated total copies sold to all buyers, sorted from the IP Item that has sold the most individual copies to the IP Item that has sold the least:

```
CREATE VIEW ITEM_TRANSACTION_IDS
SELECT I.Item_name, R.Transaction_num
FROM IP_ITEM as I, TRANSACTION_RECORDS_CONSISTOF_IP_ITEMS as R
WHERE I.Item_name = R.Item_name;

CREATE VIEW ITEM_TRANSACTIONS
SELECT I.Item_name, R.Transaction_num
FROM ITEM_TRANSACTION_IDS as I, TRANSACTION_RECORD as R
WHERE I.Transaction_num = R.Transaction_num;

CREATE VIEW IP_ITEM_SOLD_COPIES
SELECT Item_name, count(distinct Item_name)
FROM ITEM_TRANSACTIONS
ORDER BY 2 DESC;
```

To provide a list of the IP Item names and associated dollar totals for copies sold to all buyers, sorted from the IP Item that has sold the highest dollar amount to the IP Item that has sold the smallest:

```
CREATE VIEW ITEM_TRANSACTION_IDS
SELECT I.Item_name, I.Price, R.Transaction_num
FROM IP_ITEM as I, TRANSACTION_RECORDS_CONSISTOF_IP_ITEMS as R
WHERE I.Item_name = R.Item_name;

CREATE VIEW ITEM_TRANSACTIONS
SELECT I.Item_name, I.Price, R.Transaction_num
FROM ITEM_TRANSACTION_IDS as I, TRANSACTION_RECORD as R
WHERE I.Transaction_num = R.Transaction_num;

SELECT Item_name, sum(Price)
```

```
FROM ITEM_TRANSACTIONS
GROUP BY Item_name;
```

To find the most popular Seller (i.e. the one who has sold the most IP Items):

```
CREATE VIEW STORE_ITEMS(Seller_Email, Item_count)
SELECT a.Seller_email, COUNT(b.Item_name)
FROM SELL_ACCT_HOSTS_INV AS a, VIRTUAL_INVENTORY AS b,
INV_CONSISTSOF_IP_ITEMS as c
WHERE a.Store_name = b.Store_name AND b.Store_name = c.Store_name;

CREATE VIEW SELLER_ITEMS(Seller_email, Total_items)
SELECT COUNT(b.Item_count)
FROM SELLER_ACCOUNT AS a, STORE_ITEMS AS b
WHERE a.Email = b.Seller_email;

SELECT a.Name
FROM SELLER_ACCOUNT AS a, SELLER_ITEMS AS b
WHERE MAX(b.Total_items)
AND b.Seller_email = a.Email;
```

To find the most profitable seller (i.e. the one who has brought in the most money)

```
CREATE VIEW STORE_REVENUE(Seller_email, Store_money)
SELECT a.Store_name, SUM(b.Price)
FROM SELL_ACCT_HOSTS_INV AS a, VIRTUAL_INVENTORY AS b,
INV_CONSISTSOF_IP_ITEMS as c
WHERE a.Store_name = b.Store_name AND b.Store_name = c.Store_name;

CREATE VIEW SELLER_REVENUE(Seller_email, Total_revenue)
SELECT SUM(b.Store_money)
FROM SELLER_ACCOUNT AS a, STORE_REVENUE AS b
WHERE a.Email = b.Seller_email;

SELECT a.Name
FROM SELLER_ACCOUNT AS a, SELLER_REVENUE AS b
WHERE MAX(b.Total_revenue)
AND b.Seller_email = a.Email;
```

To provide a list of buyer names for buyers who purchased anything listed by the most profitable Seller:

Let the given Seller be identified by the unique email ‘smithybob@rocketmail.com’ using the query above

```
CREATE VIEW SELLER_INV
```

```
SELECT VI.Store_name
FROM SELL_ACCT_HOSTS_INV AS SAHI, VIRTUAL_INVENTORY AS VI
WHERE SAHI.Store_name = VI.Store_name AND SAHI.Seller_email = "smithybob@rocketmail.com";

CREATE VIEW INV_ITEMS
SELECT II.Item_name
FROM SELLER_INV AS SI, INV_CONSISTSOF_IP_ITEMS AS ICII, IP_ITEM as II
WHERE SI.Store_name = ICII.Store_name AND II.Item_name = ICII.Item_name

CREATE VIEW TRANSACTIONS
SELECT TR.Transaction_num
FROM INV_ITEMS AS II, TRANSACTION_RECORDS_CONSISTOF_IP_ITEMS AS CO,
TRANSACTION_RECORDS AS TR
WHERE CO.Item_name = II.Item_name AND CO.Transaction_num = TR.Transaction_num

SELECT B.Name
FROM TRANSACTIONS AS T, BUYER_ACCOUNT AS B,
BUYER_CONTAINS_TRANSACTION_RECORDS AS BCTR
WHERE BCTR.Transaction_num = T.Transaction_num AND BCTR.Buyer_email = B.Email
```

To provide the list of sellers who listed the IP Items purchased by the buyers who have spent more than the average buyer:

```
CREATE VIEW BUYER_TRANSACTIONS
SELECT BCTR.Buyer_email, BCTR.Transaction_num, TR.Store_name
FROM TRANSACTION_RECORDS AS TR, BUYER_CONTAINS_TRANSACTION_RECORDS AS
BCTR
WHERE BCTR.Transaction_num = TR.Transaction_num

CREATE VIEW TRANSACTION_ITEMS
SELECT BT.Buyer_email, BT.Store_num, II.Price
FROM BUYER_TRANSACTIONS AS BT, TRANSACTION_RECORDS_CONSISTOF_IP_ITEMS
AS CO, IP_ITEM AS II,
WHERE CO.Transaction_num = BT.Transaction_num AND CO.Item_name = II.Item_name

CREATE VIEW BUYER_TOTALS
SELECT Buyer_email, Store_num, sum(Price)
FROM TRANSACTION_ITEMS
GROUP BY Buyer_email

CREATE VIEW BUYERS_HIGH_SPENDING
SELECT Buyer_email, Store_num
FROM BUYER_TOTALS
WHERE Price > avg(price)

SELECT S.Name
```

```
FROM SELL_ACCT_HOSTS_INV AS SAHI, BUYERS_HIGH_SPENDING AS BHS,
SELLER_ACCOUNT AS S
WHERE SAHI.Store_name = BHS.Store_num AND SAHI.Seller_email = S.Email
```

Section III - Graded Checkpoint Documents

Please see additional files or click on the link below for the graded checkpoints.

Section III - Graded Checkpoint Documents

Part II

Please see additional files.